Algorithm to calculate the Hausdorff Distance on sets of points represented by k^2-tree

Fernando Domínguez
Mag. en Cs. de la Computación
Universidad del Bío-Bío
Chillán, Chile
fdomingu@egresados.ubiobio.cl

Gilberto Gutiérrez
Dep. de Cs. de la Computación y TI
Universidad del Bío-Bío
Chillán, Chile
ggutierr@ubiobio.cl

Miguel Romero
Dep. de Cs. de la Computación y TI
Universidad del Bío-Bío
Chillán, Chile
miguel.romero@ubiobio.cl

***Resumen*—The Hausdorff distance between two sets of points A and B corresponds to the largest of the distances between each object $x \in A$ and its nearest neighbor in B. The Hausdorff distance has several applications, such as comparing medical images or comparing two transport routes. There are different algorithms to compute the Hausdroff distance, some operate with the sets of points in main memory and others in secondary memory. On the other hand, to face the challenge of indexing large sets of points in main memory, there are compact data structures such as k^2-tree which, by minimizing storage, can be efficiently consulted. An efficient algorithm (HDK2) that allows the calculation of the Hausdorff distance in the compact structure k^2-tree is presented in this article. This algorithm achieves an efficient solution in both time and space. Through a series of experiments, the performance of our algorithm was evaluated together with others proposed in literature under similar conditions. The results allow to conclude that HDK2 has a better performance in runtime than such algorithms.**

***Index Terms*—Algorithms, Compact Data Structures, Hausdorff distance, k^2-tree**

I. Introducción

El cálculo de la distancia de Hausdorff es un problema antiguo, tratado especialmente en el ámbito de la topología [1]. En Ciencia de la Computación ha sido tratado a través de los últimos treinta años. En efecto, el primer trabajo encontrado data del año 1983 donde los autores [2] proponen un algoritmo para el cálculo de distancia de Hausdorff entre polígonos convexos.

La distancia de Hausdorff es principalmente una medida de similitud entre dos conjuntos de puntos A y B y, por lo tanto, puede ser utilizada para comparar formas geométricas tales como polilíneas, polígonos convexos [2] o poliedros generales representados como mallas triangulares [3]. Esto tiene muchas aplicaciones prácticas debido a que es posible modelar varias situaciones del mundo real como un conjunto de puntos o bien como figuras geométricas. Por ejemplo, esta medida, es usada en el área de la topología para la comparación de planos [1]. También es utilizada para comparar imágenes, por ejemplo es utilizada en el área de la dermatología para identificar la similitud entre imágenes de lesiones cutáneas de un paciente y otro [4]; o utilizada para conocer la evolución del tamaño de un tumor cerebral a través del tiempo [4]; y para el reconocimiento facial [5]–[7].

Formalmente, la distancia de Hausdorff es una función $\mathsf{HausDist}(A, B)$ donde A y $B \subseteq \mathbb{R}^d$, definida como [8]:

$$\mathsf{HausDist}(A, B) = \max_{x \in A}\{\min_{y \in B}\{\mathsf{Dist}(x, y)\}\}$$

es decir, es la mayor de las distancias entre cada punto $x \in A$ con su vecino más cercano $y \in B$. $\mathsf{HausDist}(A, B)$ también es llamada Distancia directa de Hausdorff. Notar que la Distancia de Hausdorff es antisimétrica ($\mathsf{HausDist}(A, B) \neq \mathsf{HausDist}(B, A)$). La Distancia simétrica de Hausdorff se define como [9]:

$$\mathsf{SymHD}(A, B) = \max\{\mathsf{HausDist}(A, B), \mathsf{HausDist}(B, A)\}$$

En la Figura 1 se presentan dos trayectorias[1] A y B, en este caso la distancia de Hausdorff se puede considerar como la mayor discrepancia de una trayectoria con otra, donde la noción de discrepancia entre trayectorias es una medida de similitud [10]. De este modo la mayor discrepancia de A con respecto a B es $\mathsf{HausDist}(A, B) = \mathsf{Dist}(a_4, b_4)$ y la mayor discrepancia de B con respecto a A es $\mathsf{HausDist}(B, A) = \mathsf{Dist}(b_5, a_4)$. Una aplicación práctica de lo anterior, sería por ejemplo, identificar cual es la máxima distancia adicional que tendría que recorrer un pasajero si la autoridad de transporte decidiera reemplazar una ruta de bus (secuencia de paradas) por otra [9], de este modo, la autoridad de transporte podría evaluar diferentes rutas alternativas con la finalidad de escoger la que produzca el menor impacto en la población.

Por otro lado, con el creciente aumento en los volúmenes de información, se hace indispensable utilizar eficientemente el almacenamiento para la representación de los datos. El tamaño de los datos, no solo impacta en el costo de almacenamiento, también influye en el costo de transmisión de los datos y en su procesamiento.

Un enfoque interesante para abordar este problema emerge del campo de los sistemas de recuperación de información, las llamadas estructuras de datos compactas [11]. Estas estructuras de datos permiten representar la información de una manera eficiente, utilizando un espacio muy cercano al óptimo teórico según la teoría de la información. Junto con ello, las estructuras compactas mantienen las propiedades de acceso directo

[1]Una trayectoria es la secuencia de puntos por los cuales a pasado un objeto móvil. La secuencia de puntos se encuentra ordenada por el tiempo

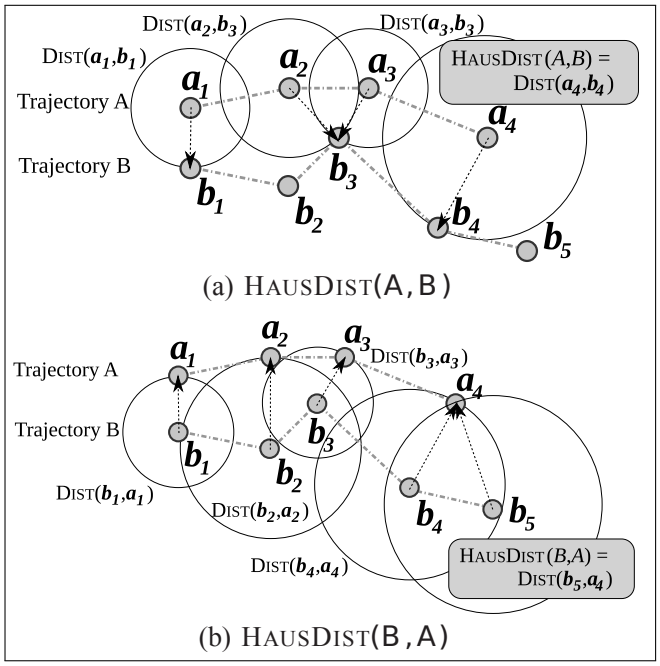

Figura 1: Ejemplo de cálculo de distancia de Hausdorff, tanto de A a B como de B a A [9].

a los datos, lo que permite realizar operaciones sobre ellos directamente, sin descompactar toda la estructura.

En los ejemplos descritos anteriormente es fácil visualizar contextos en los cuales existan grandes volúmenes de puntos, por ejemplo en la comparación de imágenes de alta resolución. Dichas imágenes son más costosas de analizar cuando requieren la indexación de los puntos en memoria secundaria que cuando están en memoria principal, simplemente por la localidad de los datos.

En la literatura existen varias estructuras de datos compactas para representar conjuntos de puntos, entre ella el Wavelet Tree [12] y el k^2-tree [13]. El Wavelet Tree destaca porque permite obtener un almacenamiento teórico óptimo, a diferencia del k^2-tree. Sin embargo, en la práctica con el k^2-tree se puede obtener un menor costo de almacenamiento que con Wavelet Tree cuando los puntos se encuentran agrupados [11]. De acuerdo a la literatura revisada no existe un algoritmo que calcule la distancia de Hausdorff sobre estructuras de datos compactas.

En este contexto, si alguien decide utilizar una estructura compacta para representar conjuntos de puntos tiene dos alternativas para calcular la distancia de Hausdorff i) extraer los puntos desde la estructura compacta y luego procesarlo con algún algortimo conocido que proceso los datos sin indexarlos previamente; y (ii) calcular la distancia de Hausdorff con un algoritmo que procese directamente sobre la estructura compacta.

En este artículo se propone un nuevo algoritmo para calcular la distancia de Hausdorff considerando que los conjuntos, son puntos de dos dimensiones y están representados mediante k^2-trees. Nuestro algoritmo sigue las ideas planteadas en [9] y [8] y [14], pero adaptándolas a la naturaleza de la estructura. En nuestro algoritmo usamos la distancia euclídea para el cálculo de la distancia de Hausdorff.

En la siguiente sección se presentan los trabajos relacionados a algoritmos que solucionan el problema del cálculo de la distancia de Hausdorff, ademas de trabajos que tratan acerca de la estructura de datos compacta k^2-tree. En la Sección 3 se describe nuestro trabajo, un nuevo algoritmo para calcular la distancia de Hausdorff. En la Sección 4 presentamos los experimentos realizados. Finalmente, las conclusiones y el trabajo futuro se presentan en la Sección 6.

II. TRABAJOS RELACIONADOS

En esta sección se discuten los algoritmos propuestos en la literatura que son utilizados para calcular la distancia de Hausdorff. Adicionalmente se describe la estructura de datos compacta k^2-tree.

II-A. Distancia de Hausdorff en memoria principal

El primer algoritmo encontrado en la literatura para calcular la distancia de Hausdorff en memoria principal es el propuesto por Atallah [2]. Este algoritmo es utilizado para calcular la distancia de Hausdorff entre dos polígonos convexos. Dicho algoritmo es de complejidad temporal $\mathcal{O}(m+n)$, donde m y n corresponden al número de vértices exteriores de cada polígono convexo. En la misma línea de comparar formas poligonales, Tang *et. al.* [15] presentan un algoritmo para calcular la distancia de Hausdorff entre modelos poligonales complejos, el cual a diferencia del algoritmo anterior no requiere condiciones especiales sobre el conjunto de puntos. El algoritmo propuesto utiliza una subdivisión de Voronoi para dividir el modelo poligonal en triángulos más pequeños para calcular la distancia de Hausdorff entre los vértices de dichos triángulos, pero el resultado es aproximado. Helmut *et. al.* en [16] presentan algoritmos que también utilizan diagrama de Voronoi para calcular la distancia de Hausdorff cuya complejidad temporal es $\mathcal{O}((n+m)\log(n+m))$.

Por otro lado, Taha *et. al.* [8] proponen un algoritmo que, basado en lo que el autor define como un *"rompimiento temprano"* del proceso, busca evitar la evaluación de todo el conjunto de puntos.

Al escanear los puntos de B con respecto a un punto de A, el rompimiento temprano ocurre al encontrar una distancia d menor a lo que ya se definió como la distancia mayor $cmax$ de las menores distancias ya determinadas. Esto ocurre por la definición implícita del cálculo de la distancia de Hausdorff la cual señala que se deben maximizar las mínimas distancias encontradas. Es por esto que si se encuentra una distancia d menor a lo que ya se definió como la distancia mayor $cmax$, no es necesario escanear el resto de punto de B. Este rompimiento temprano del algoritmo le permite mejorar su eficiencia evitando eventualmente recorrer todos los puntos de B. Cuando los puntos son obtenidos desde imágenes, el escaneo horizontal o vertical de los puntos impide que el rompimiento temprano sea efectivo. Para enfrentar este

problema Taha [8] propone randomizar los puntos antes de calcular la distancia de Hausdorff.

Otro algoritmo que utiliza esta técnica de rompimiento temprano es el expuesto por Chen *et. al.* [17], quienes presentan un algoritmo llamado *Local Start Search (LSS)*. El rompimiento temprano de este algoritmo proviene de la probabilidad de que los puntos estén en zonas de densidad. Estos rompimientos tempranos son realizados en los bucles de escaneo de puntos. Además, el algoritmo utiliza un mecanismo para registrar la posición del último rompimiento como una posición de inicio, la cual es utilizada como centro de búsqueda del siguiente bucle, explorando los puntos vecinos alrededor de este centro.

En otra línea, Guthe *et. al.* [18] proponen un algoritmo para calcular la distancia de Hausdorff para la simplificación de mallas geométricas. Este algoritmo para evitar el escaneo de todos los puntos saca ventajas de las características específicas que poseen las mallas geométricas. El algoritmo solo escanea intensivamente aquellas áreas donde existen distancias máximas entre los triángulos de cada malla. Para ello construye una grid a través de una estructura de datos Octree [19] en la cual almacena cada superficie, calculando las distancias mínimas y máximas entre ellas. Además, al igual que [9], utiliza una cola de prioridad, en la cual almacena las celdas ya procesadas, las que ordena por su distancia geométrica máxima.

Para comparar imágenes, Huttenlocher *et. al.* [20], proponen dos algoritmos, uno para objetos en $\mathbb{R}^2$ con complejidad $\mathcal{O}(mn \log mn)$ y otro algoritmo para objetos en $\mathbb{R}^3$ con complejidad $\mathcal{O}(m^2n^2 \propto (mn))$, donde m y n es el número de puntos de cada conjunto y $\propto (mn)$ corresponde a la función de traslación de dichos conjuntos. Estos algoritmos calculan la distancia de Hausdorff a través de los polígonos generados por los puntos exteriores de los conjuntos de datos, siguiendo la misma idea presentada en [2]. Utilizando este mismo método de traslación, pero específicamente para el cálculo de la distancia de Hausdorff entre dos líneas, Rote [21] y Li *et. al.* [22] proponen algoritmos que realizan algunas mejoras al algoritmo presentado en [20]. La complejidad de estos algoritmos es $\mathcal{O}((n+m)\log(n+m))$. Ahora bien, el problema de estas técnicas de traslación es que requieren mucho tiempo de cómputo dependiendo de la cantidad de puntos de cada conjunto. Para disminuir este tiempo Chang *et. al.* [23] proponen un algoritmo que utiliza un filtro de Kalman [24] para filtrar puntos y con ello mejorar el tiempo de cómputo en la comparación de dichos conjuntos.

La distancia de Hausdorff también puede ser calculada entre curvas. Un algoritmo para esto es el presentado por Chen *et. al.* [25]. Este algoritmo utiliza un método heurístico para dividir las curvas en sub-intervalos a través de curvas B-spline. Además, este algoritmo utiliza técnicas de poda geométrica para eliminar aquellos sub-intervalos que no aportan a la solución final.

II-B. Distancia de Hausdorff en memoria secundaria

Nutanong *et. al.* [9] proponen algoritmos que utilizan estructuras de datos R-tree [26], las cuales son almacenadas en memoria secundaria. El uso de estas estructuras R-tree

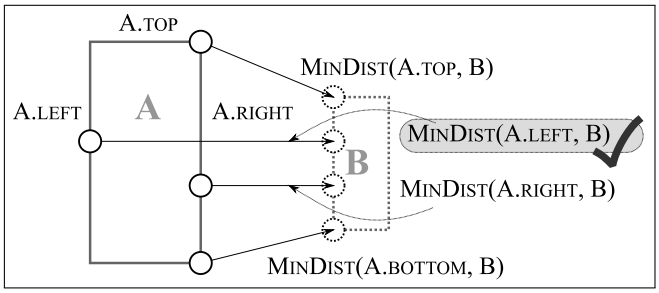

Figura 2: Límite inferior definido por Nutanong [9].

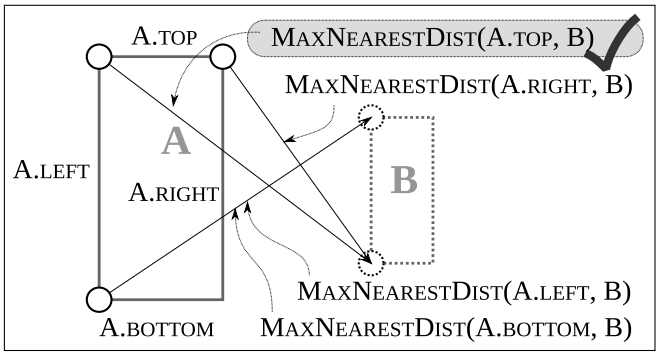

Figura 3: Límite superior definido por Nutanong. [9].

permite la indexación de los conjuntos A y B, con lo cual se puede realizar podas de aquellos puntos que no aportan a la solución. Los algoritmos presentados requieren de algunas definiciones previas. Una de ella es la definición de un límite inferior, el cual es calculado como la mínima distancia entre la cara del rectángulo generado por el MBR(Minimum Bounding Rectangle) de A que está frente a la cara del rectángulo generado por el MBR de B, esto puede ser visto en la Figura 2.

Además, los autores definen un límite superior, el cual es calculado como la máxima distancia entre la cara del MBR del conjunto A, con respecto al MBR del conjunto B, tal como lo muestra la Figura 3.

II-C. Estructuras de Datos Compactas

Las estructuras datos compactas son estructuras que permiten representar datos utilizando de forma eficiente el espacio, mientras que todavía son capaces de resolver de forma eficiente operaciones requeridas sobre los datos [11], [13]. Existen estructuras de datos compactas para: secuencias binarias (*bitmaps*), secuencias de símbolos, árboles cardinales, permutaciones, grafos, indexación de rangos, entre otras.

Entre las estructuras de datos compactas podemos encontrar el k^2-tree, ver Figura 4, el cual es definido por Brisaboa et. al [27] como una estructura de datos compacta basada en QuadTree [28]. Esta estructura también puede verse como una relación binaria representada por un árbol k^2-ario de altura $h = \lceil \log_k n \rceil$, el cual representa una matriz de adyacencia binaria de rango $n \times n$ de manera compacta. Puede ser usada por ejemplo para representar grafos de propósito general [29],

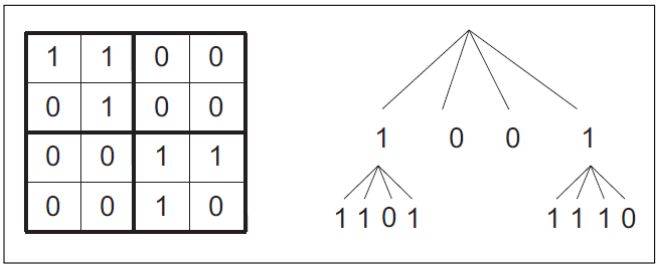

Figura 4: Representación de una matriz de adyacencia en un k^2-tree [27].

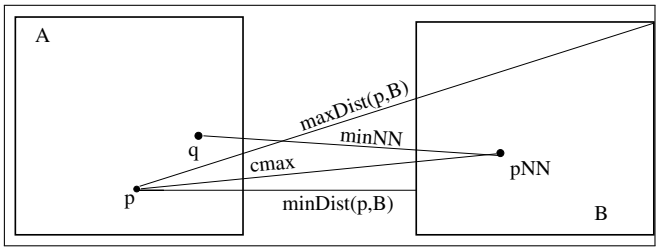

Figura 5: Métricas utilizadas en las reglas de poda

datos RDF (Resource Description Framework), o un grid bi-dimensional de puntos [30].

Con respecto a la estructura del árbol conceptual (Figura 4) debemos decir que cada nodo del árbol está etiquetado con un bit 1 si la sub-matriz correspondiente contiene al menos un uno, o con un bit 0 si la sub-matriz está llena de ceros. Así el proceso continúa recursivamente sólo con aquellas sub-matrices que contienen al menos un 1. La subdivisión recursiva de la matriz se detiene cuando la sub-matriz actual está llena de ceros o cuando se completan las celdas de la matriz de adyacencia original. Esto puede verse en la figura 4. Finalmente, lo que se almacena en el k^2-tree es la secuencia de bits resultantes de recorrer el árbol conceptual en anchura, guardando separadamente el último nivel (*bitmap* L) del resto del árbol (*bitmap* T). Para la figura 4, los *bitmaps* que son almacenados son $T = 1001$ y $L = 11011110$.

El ahorro de espacio en un k^2-tree, está dado por la existencia de cuadrantes que se encuentran llenos de ceros y, por lo tanto, no son subdivididos. Además, no se almacena explícitamente la topología del árbol, ahorrando el espacio requerido por los punteros y nodos de una representación clásica.

Algunas operaciones que se pueden realizar en un k^2-tree, son la recuperación de los vecinos directos y reversos, recuperaciones de celdas, consultas por rango.

Una variante del k^2-tree es el enfoque Híbrido, en el cual busca mejorar los tiempos de consulta. Para ello mantiene dos valores de k, uno valor de k más grande para los niveles superiores, y uno valor de k menor para los niveles inferiores, con esto se logra disminuir la altura del árbol.

Otra versión especial del k^2-tree es aquella que utiliza compresión de unos. Esta estructura busca obtener una representación eficiente para matrices binarias que contienen grandes grupos de ceros y unos. Un ejemplo de estas matrices son aquellas que representen imágenes binarias. La principal diferencia con el k^2-tree original es la filosofía de división de las sub-matrices. En esta estructura la descomposición original se detendrá cuando se encuentren regiones uniformes, ya sea de ceros o unos. Esto tiene como ventaja reducir el número de nodos en el árbol conceptual cuando grandes regiones de unos aparecen en la matriz binaria.

Otra variante es el Dk^2-tree el cual es una versión dinámica del k^2-tree. En esta estructura al cambiar una celda de la matriz de adyacencia dinámicamente se hace la modificación en el árbol. Esto permite hacer modificaciones posteriores a la construcción inicial del árbol.

El almacenamiento requerido por un k^2-tree para almacenar un conjunto de puntos ha sido estudiado en diversos trabajos tales como en [14], [30], [31] los que en general concluyen que el k^2-tree es eficiente para representar puntos, en especial si los puntos se encuentran concentrados en regiones del espacio.

III. Algoritmo HDK2

Nuestro algoritmo asume que los conjuntos de puntos A y B se encuentran almacenados en los k^2-trees K_A y K_B respectivamente. El algoritmo explora exhaustivamente los puntos en K_A y por cada uno de ellos encuentra su vecino más cercano en K_B, siempre y cuando dicho vecino sea una posible solución.

La estrategia seguida por el algoritmo es de ramificación y poda. Concretamente usa tres reglas de poda basadas en la distancia euclídea entre dos puntos y de un punto a un rectángulo.

El algoritmo hace uso de las funciones $Dist(p_1, p_2)$ para obtener la distancia entre dos puntos y las función $minDist(p, B)$ y $maxDist(p, B)$ para obtener la menor y la mayor distancia entre el punto p y el rectángulo B respectivamente (ver Fig. 5).

La función $minDist(.,.)$ se encuentra formalmente definida en [32]. Por su parte la función $maxDist(.,.)$ corresponde a una caso especial de la función $MAXMAXDIST(S, R)$ con R y S dos rectángulos definida en [33] en que S o R es un rectángulo definido por solo un punto y corresponde a q.

A continuación se explicará en detalle en que consisten las tres podas que utiliza nuestro algoritmo.

III-A. Reglas de poda

Dado un punto $p \in A$, los valores de las funciones $minDist(p, B)$ y $maxDist(p, B)$ representan los límites inferior y superior respectivamente del intervalo en el cual se encuentra la distancia del vecino más cercano de p dentro del rectángulo B.

Asumimos que $cmax$ representa una solución parcial (solución candidata) de la distancia de Hausdorff y cuyo valor se obtuvo entre $Dist(p, pNN)$ (ver Fig. 5) con $p \in A$ y $pNN \in B$. Las reglas son las siguientes:

1. **Regla de poda 1:** Si durante la exploración de K_B se encuentra un punto $pNN \in B$ tal que $minNN =$

$Dist(q, pNN) \leq cmax$, con $q \in A$, entonces ya no es necesario seguir explorando K_B, pues el vecino más cercano de q no mejorará la solución mantenida en $cmax$, ya que si $t \neq pNN \in B$ es el vecino más cercano de q, entonces $Dist(q, t) \leq Dist(q, pNN)$.

2. **Regla de poda 2:** Esta segunda regla se basa en la distancia $maxDist(.,.)$ (ver Fig. 5). Considere que está buscando en K_B el vecino más cercano para un punto $q \in A$, y R es la región del espacio representada por K_B, si durante la búsqueda ocurre que $maxDist(q, R) \leq cmax$, entonces ya no es necesario seguir explorando K_B, pues el vecino más cercano de q se encontrará a una distancia menor o igual que $maxDist(q, R)$ y por lo tanto el valor de la distancia de Hausdorff mantenida hasta el momento (valor de $cmax$) no cambiará.
3. **Regla de poda 3:** Adicional a las dos reglas anteriores, el algoritmo utiliza la regla de poda basada en $minDist(.,.)$ y que típicamente usan los algoritmos conocidos (por ejemplo el propuesto en [32]) para calcular el vecino más cercano. Dicha regla descarta regiones o zonas delimitadas por un rectángulo R generadas por K_B. Específicamente, si $q \in A$, $minNN$ la mejor distancia conocida para el vecino más cercano de q y $minDist(q, R) > minNN$, entonces la región R no necesita ser explorada.

III-B. Descripción del algoritmo HDK2

En Alg. 1 se muestra la parte principal del algoritmo HDK2. Se asume que la función $getRoot(.)$ obtiene el nodo raíz del k^2-tree (línea 3). Esencialmente el tamaño de la matriz y el nivel del nodo. De la misma manera $getRegion(.)$ permite obtener la región del espacio cubierta por K_B representada mediante un rectángulo definido por dos puntos extremos opuestos (línea 4). Tal como indicamos en la sección anterior, $cmax$ es una variable utilizada para mantener la solución de la distancia de Hausdorff, la cual se maximiza conforme el algoritmo progresa. Luego HDK2 por cada punto $p \in A$ almacenado en K_A verifica, mediante el algoritmo $NNMax$ (ver Alg. 2), si la distancia entre p y su vecino más cercano mejora la distancia de Hausdorff mantenida en $cmax$ (líneas 5-10 de Alg. 1). Notar que en la línea 7 se aplica la regla de poda 1. Se asume que $pNN \in B$ es un punto que optimizó la variable $cmax$ para un punto en A procesado antes que el punto $p \in A$. En esta etapa del algoritmo, la distancia entre p y pNN ($Dist(p, pNN)$) constituye una cota superior para el vecino más cercano de p; y por lo tanto se ejecuta el algoritmo $NNMax$ solamente si la distancia entre p y su vecino más cercano mejora la solución mantenida en $cmax$. Para mayor claridad de la aplicación de la regla de poda 1 podemos ver la Fig. 5 en donde asumiendo que p fue procesado antes que q, es claro que si $minNN = Dist(q, pNN) \leq cmax$ la distancia entre q y pNN no mejorará el valor de $cmax$.

El algoritmo $NNMax$ (ver Alg. 2), en general permite calcular el vecino más cercano de un punto $q \in A$ sobre el conjunto de puntos B representado por el k^2-tree K_B; con el propósito de mejorar la distancia de Hausdorff. Eventualmente $NNMax$, mediante las reglas de poda definidas previamente puede evitar calcular el vecino más cercano de q y decidir tempranamente que la distancia entre q y su vecino más cercano no mejorará la solución.

$NNMax$ usa una cola de prioridad representada por un heap pQ (ver Alg. 2, línea 1). Cada entrada de pQ tiene la estructura $\langle n, r, d\rangle$ con n el nodo del k^2-tree, r la región del nodo n y d la distancia máxima ($maxDist(q, r)$) entre q y r. pQ está organizado por d, es decir, el menor valor de d se encuentra en la raíz de la heap.

Alg. 1 Algoritmo HDK2 para calcular la distancia de Hausdorff sobre dos conjuntos de puntos almacenados en k^2-tree

HDK2(K_A, K_B)
input: Dos k^2-Tree K_A y K_B.
output: La distancia de Hausdorff.

```
1:  pNN = (∞, ∞)
2:  cmax = 0                          ▷ cmax distancia de Hausdorff
3:  root =GETROOT(K_B)
4:  r = GETREGION(Node)
5:  for Cada punto p en K_A do
6:      minNN = Dist(p, pNN)
7:      if minNN > cmax then                          ▷ regla 1
8:          cmax = NNMAX(root, r, p, cmax, minNN)
9:      end if
10: end for
11: return cmax
```

El algoritmo $ANNMax$ (Alg. 2) inicialmente (línea 3) crea e inserta una entrada en la cola de prioridad pQ. Tal entrada considera la raíz de K_B, la región representada por el K_B y la distancia ($maxDist(.,.)$) de q a dicha región. Luego el algoritmo ejecuta un ciclo (líneas 4-29) procesando las entradas de pQ para calcular el vecino más cercano de q que eventualmente mejorará la solución. La función $Extract - Min(.)$ (línea 5) recupera y elimina la entrada de pQ con el menor valor de d.

En las líneas 6-13 se procesan las entradas e de pQ que corresponden a puntos de B los que se encuentran en las hojas de K_B y que se ubican en el último nivel. Los nodos hojas que no se encuentran en el último nivel representan regiones en las cuales no existen puntos (zonas vacías) y por lo tanto son descartadas por el algoritmo. En este tipo de entradas $e.r$ representa las coordenadas del punto de B almacenado en la hoja y por lo tanto $e.d$ ($maxDist(q, r)$) corresponde a la distancia entre los puntos q y $e.r$. Notar que en las líneas 7-9 se aplica la regla de poda 1. En este punto del algoritmo, como $e.d$ representa una distancia real entre q y $e.r$ y si dicha distancia es menor o igual que la candidata en $cmax$, entonces ya no es necesario continuar explorando K_B pues el vecino más cercano real de q será menor o igual a $e.d$ y por lo tanto ya no mejorará el valor de $cmax$, y el algoritmo termina.

Entre las líneas 15-28 se procesan las entradas e de pQ correspondientes a nodos intermedios (no hojas) de K_B. Dicho procesamiento consiste básicamente en expandir el nodo $e.n$ de K_B e insertar los nodos hijos no vacíos de $e.n$ en pQ.

Alg. 2 (**NNMAX**)Maximización de la Distancia de Hausdorff

NNMAX($rootNode, Region, q, cmax, minNN$)
input: Nodo raíz de k²-tree K_B, el punto $q \in A$, la distancia de Hausdorff $cmax$ candidata y $minNN$ la cota superior para la distancia entre q y su vecino más cercano.
output: La distancia de Hausdorff $cmax$ actualizada.

```
1:  pQ =CREATEMIN-HEAP( )
2:  d = MAXDIST(q, Region)
3:  INSERT(pQ, ⟨rootNode, Region, d⟩)
4:  while (EMPTY(pQ) ≠ ∅) do
5:      e = EXTRACT-MIN(pQ)
6:      if ISLEAF(e.n) then
7:          if (e.d ≤ cmax) then                    ▷ regla 1
8:              return cmax
9:          end if
10:         if e.d < minNN then
11:             minNN = e.d
12:             pNN1 = e.r
13:         end if
14:     else
15:         for all Nodo h hijo de e.n do
16:             if (hasChildren(h) then
17:                 r = GETREGION(h)
18:                 mayDist = MAXDIST(q, e.r)
19:                 if (mayDist ≤ cmax) then        ▷ regla 2
20:                     return cmax
21:                 end if
22:                 menDist = MINDIST(q, e.r)
23:                 if (menDist < minNN) then       ▷ regla 3
24:                     INSERT(pQ, ⟨h, r, mayorDis⟩)
25:                 end if
26:             end if
27:         end for
28:     end if
29: end while
30: pNN = pNN1
31: return minNN
```

Sobre estos nodos se aplica la regla de poda 2 (líneas 19-21). Para ello se calcula la cota superior para el vecino más cercano de q ($maxDist(q, r)$) y que se encuentra dentro de la región r de h. Es claro que si dicha cota está por debajo de $cmax$ el vecino más cercano de q en B no mejorará la solución, y por lo tanto ya no es necesario seguir explorando K_B, el algoritmo termina.

Antes de insertar una entrada en pQ se aplica la regla 3. Es decir, se calcula la cota inferior de la distancia del vecino más cercano de q en r ($minDist(q, r)$). Si el valor de esta cota está por sobre $minNN$, es decir, la distancia candidata entre q y su vecino más cercano descubierto hasta este punto del algoritmo, entonces la entrada correspondiente al hijo h de $e.n$ no se agrega a la cola de prioridad pQ, pues no minimizará el valor de $minNN$.

IV. EVALUACIÓN EXPERIMENTAL

Con el propósito de evaluar nuestro algoritmo HDK2 se realizaron una serie de experimentos que permiten compararlo contra el algoritmo propuesto por [8] (que identificamos como Taha) y contra un algoritmo ingenuo (que denotamos como Naive). Hacemos notar que tanto para el algoritmo Taha y Naive es necesario, previamente, recuperar los puntos desde los k²-trees en arreglos diferentes, y a partir de allí calcular la distancia de Hausdorff. Notar que esta situación es realista, pues al no existir un algoritmo que resuelva el problema de la distancia de Hausdorff directamente sobre los k²-tree, Taha es la opción más directa. La diferencia entre Taha y Naive es que este último no utiliza ninguna estrategia de poda y por tanto realiza una búsqueda exhaustiva de cada uno de los vecinos más cercanos. El algoritmo Naive tiene por objetivo solo servir de referencia para comparar los algoritmos Taha y HKD2. Los tres algoritmos fueron implementado en el lenguaje de programación C.

Los experimentos fueron realizados en un computador con procesador Intel(R) Xeon(R) CPU E3-1220 V2 @ 3.10GHz, Memoria RAM 23 GB y Sistema Operativo Linux Debian 3.2. En los experimentos medimos el rendimiento mediante el tiempo de ejecución.

Se realizaron experimentos con puntos generados de manera aleatoria siguiendo las distribuciones uniforme y gaussiana dentro de un espacio 2D de rango $2^{16} \times 2^{16}$. Los tamaños de los conjuntos fueron 1.000, 10.000, 100.000 y 1.000.000 de puntos. Para analizar el tiempo de ejecución, por cada tamaño de conjunto se generaron seís pares de conjuntos y se calculó el tiempo promedio utilizado para calcular la distancia de Hausdorff por cada algoritmo. El escenario para realizar los experimentos consideró que los conjuntos de puntos se encuentran almacenados en dos k²-trees. El tiempo promedio calculado incluye el tiempo necesario para extraer los puntos desde los k²-trees y el requerido para calcular la distancia de Hausdorff .

En las figuras 6 y 8 podemos ver el comportamiento de los tres algoritmos para diferentes tamaños de los conjuntos considerando distribución uniforme y gaussiana respectivamente. Podemos notar que cuando se trata de conjuntos pequeños, menos de 10.000 puntos, no existe mayor diferencia entre los algoritmos Naive, Taha y HDK2 en ambas distribuciones. Sin embargo, para los conjuntos con distribución uniforme, a partir de los 100.000 puntos, se puede ver que el algoritmo HDK2 presenta un mejor rendimiento que los otros dos, llegando a superarlos por tres órdenes de magnitud (ver Fig. 6). Al considerar conjuntos con distribución gaussiana (Fig. 8) también es posible apreciar una ventaja de HDK2 por sobre los otros dos algoritmos aunque esta es menor que la que se produce en el escenario con distribución uniforme. Es posible ver en la Fig. 8 que HDK2 supera a Taha en menos de un orden de magnitud, y no en tres como sucede con los conjuntos con distribución uniforme. La diferencia anterior se debe al agrupamiento de los puntos, lo que hace difícil que el algoritmo HDK2 utilice tempranamente las reglas de

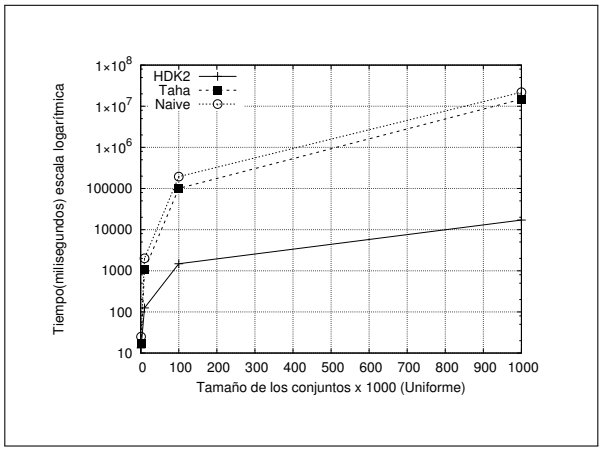

Figura 6: Tiempos de ejecución de los tres algoritmos para diferentes conjuntos de datos (distribución uniforme)

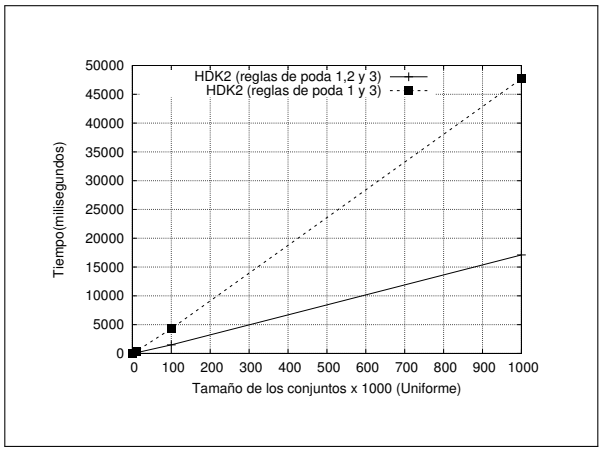

Figura 7: Tiempo de ejecución de HDK2 con y sin regla de poda 2 (distribución uniforme)

poda. Cabe destacar que la distribución de los puntos afecta el rendimiento del algoritmo HDK2 y Taha, obteniéndose un mejor rendimiento en tiempo de ejecución cuando los puntos están bajo una distribución gaussiana.

Con el propósito de evaluar el efecto de las reglas de poda en nuestro algoritmo HDK2 (ver Sección III) se realizó una serie de experimentos en dos escenarios. El primero (reglas de poda 1, 2 y 3) consistió en ejecutar nuestro algoritmo incluyendo todas las reglas de poda; y el segundo (reglas de poda 1 y 3) consistió en ejecutar HDK2 sin la regla 2. En las figuras 7 y 9 podemos ver el comportamiento de HDK2 en ambos escenarios para diferentes tamaños de los conjuntos considerando distribución uniforme y gaussiana respectivamente. Podemos notar que el uso de la poda 2 tiene un importante impacto en el tiempo de ejecución en ambas distribuciones, Ahora bien, al comparar el impacto de la poda 2 entre ambas distribuciones podemos notar que bajo una distribución gaussiana la poda 2 presenta un mayor impacto en la disminución del tiempo de ejecución.

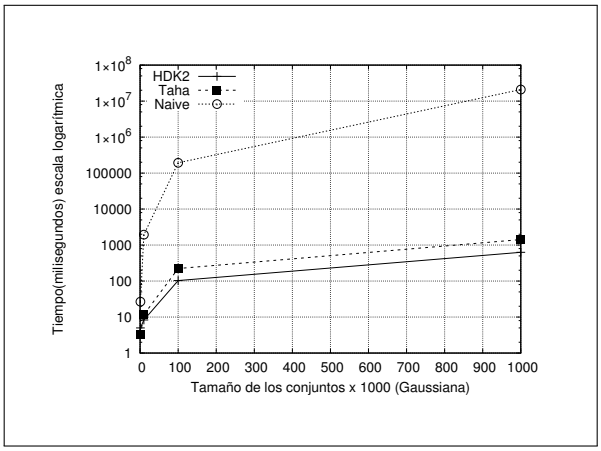

Figura 8: Tiempo de ejecución de los tres algoritmos para diferentes conjuntos de datos (distribución gaussiana)

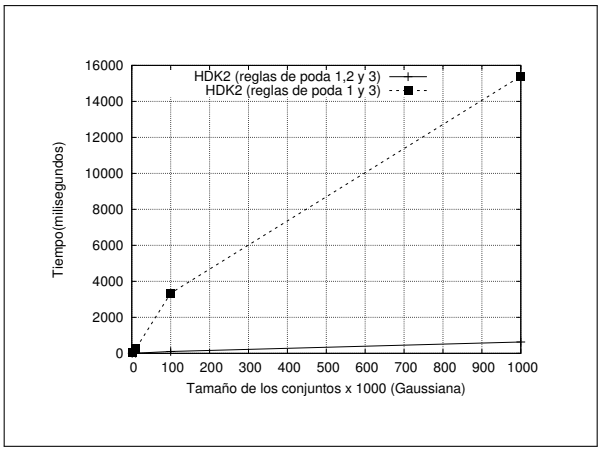

Figura 9: Tiempo de ejecución de HDK2 con y sin regla de poda 2 (distribución gaussiana)

V. Conclusión y Trabajo Futuro

En este trabajo se presenta un nuevo algoritmo (HDK2) para calcular la distancia de Hausdorff el cual asume que cada conjunto de puntos se encuentra representado en la estructura de datos compacta k^2-tree. Para calcular la distancia de Hausdorff $\mathsf{HausDist}(A, B)$ el algoritmo por cada punto de A, eventualmente, calcula la distancia de su vecino más cercano en B. Para evitar procesar todos los puntos de B el algoritmo usa tres reglas de poda las que permiten reducir el tiempo de ejecución. De acuerdo a la revisión del estado del arte y a nuestro propio conocimiento se puede concluir que actualmente no existe en la literatura un algoritmo para calcular la distancia de Hausdorff considerando que los puntos están almacenados en un k^2-tree.

Con el propósito de evaluar el rendimiento de HDK2 se realizaron una serie de experimentos los que consideraron datos sintéticos con distribución uniforme y gaussiana. Nuestro algoritmo se comparó con dos algoritmos, a saber: algoritmo de Taha [8] (Taha) y algoritmo ingenuo (Naive). Los resultados experimentales permiten concluir que, a partir de 10.000

puntos, nuestro algoritmo supera a los algoritmos de Taha y el Naive en tiempo de ejecución entre 1 y 3 órdenes de magnitud. Esta diferencia en el rendimiento del algoritmo esta determinada por la distribución de los puntos, obteniéndose un mayor diferencia en el tiempo de ejecución con respecto a los algoritmos Naive y de Taha bajo una distribución uniforme. Cabe destacar que los algoritmos HDK2 y Taha tienen un mejor rendimiento en tiempo de ejecución bajo una distribución gaussiana de puntos. También se evaluó el efecto de la regla de poda 2 sobre el tiempo de ejecución del HDK2. Los experimentos mostraron que tal regla permiten reducir el tiempo de ejecución de HDK2 a menos de la mitad para el caso de distribución uniforme y a cerca de un 0,6 % para distribución gaussiana para conjuntos de tamaño por sobre 1.000.000 de puntos.

Como trabajo futuro nos planteamos diseñar un algoritmo que permita procesar ambos conjuntos de puntos sin la necesidad extraerlos previamente de ninguno de los dos k^2-trees. También pretendemos medir, mediante experimentos, el rendimiento de nuestro algoritmo en escenarios reales y que ilustren su aplicabilidad. Finalmente, pretendemos diseñar un algoritmo para calcular la distancia de Hausdorff simétrica, es decir, $\mathsf{SymHausDist}(A, B)$.

AGRADECIMIENTOS

Gilberto Gutiérrez. Parcialmente financiado por los proyectos de investigación interno 171319 4/R y 181315 3/R de la Universidad del Bío-Bío (Chile).

Miguel Romero. Parcialmente financiado por el proyecto de iniciación DIUBB 163319 3/I de la Universidad del Bío-Bío.

REFERENCIAS

[1] J. L. Kelley, *General topology*. Springer Science & Business Media, 1975, vol. 27.

[2] M. J. Atallah, "A linear time algorithm for the hausdorff distance between convex polygons," *Information processing letters*, vol. 17, no. 4, pp. 207–209, 1983.

[3] M. Bartoň, I. Hanniel, G. Elber, and M.-S. Kim, "Precise hausdorff distance computation between polygonal meshes," *Computer Aided Geometric Design*, vol. 27, no. 8, pp. 580–591, 2010.

[4] P. Spyridonos, G. Gaitanis, I. D. Bassukas, and M. Tzaphlidou, "Gray hausdorff distance measure for medical image comparison in dermatology: Evaluation of treatment effectiveness by image similarity," *Skin Research and Technology*, vol. 19, no. 1, 2013.

[5] B. Takacs, "Comparing face images using the modified hausdorff distance," *Pattern recognition*, vol. 31, no. 12, pp. 1873–1881, 1998.

[6] Y. Gao, "Efficiently comparing face images using a modified hausdorff distance," *IEE Proceedings-Vision, Image and Signal Processing*, vol. 150, no. 6, pp. 346–350, 2003.

[7] O. Jesorsky, K. J. Kirchberg, and R. W. Frischholz, "Robust face detection using the hausdorff distance," in *International Conference on Audio-and Video-Based Biometric Person Authentication*. Springer, 2001, pp. 90–95.

[8] A. A. Taha and A. Hanbury, "An efficient algorithm for calculating the exact hausdorff distance," *IEEE transactions on pattern analysis and machine intelligence*, vol. 37, no. 11, pp. 2153–2163, 2015.

[9] S. Nutanong, E. H. Jacox, and H. Samet, "An incremental hausdorff distance calculation algorithm," *Proceedings of the VLDB Endowment*, vol. 4, no. 8, pp. 506–517, 2011.

[10] G. Trajcevski, H. Ding, P. Scheuermann, R. Tamassia, and D. Vaccaro, "Dynamics-aware similarity of moving objects trajectories," in *Proceedings of the 15th annual ACM international symposium on Advances in geographic information systems*. ACM, 2007, p. 11.

[11] G. Navarro, *Compact Data Structures: A Practical Approach*, 1st ed. New York, NY, USA: Cambridge University Press, 2016.

[12] G. Navarro., "Wavelet trees for all," *Journal of Discrete Algorithms*, vol. 25, pp. 2 – 20, 2014, 23rd Annual Symposium on Combinatorial Pattern Matching. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1570866713000610

[13] S. Ladra González, "Algorithms and compressed data structures for information retrieval," Ph.D. dissertation, Universidade da Coruna, 2011.

[14] M. Romero, "Un índice espacio temporal para puntos móviles basado en estructuras de datos compactas," Ph.D. dissertation, Universidade da Coruña, 2017. [Online]. Available: http://hdl.handle.net/2183/19303

[15] M. Tang, M. Lee, and Y. J. Kim, "Interactive hausdorff distance computation for general polygonal models," *ACM Transactions on Graphics (TOG)*, vol. 28, no. 3, p. 74, 2009.

[16] H. Alt, B. Behrends, and J. Blömer, "Approximate matching of polygonal shapes," in *Proceedings of the seventh annual symposium on Computational geometry*. ACM, 1991, pp. 186–193.

[17] Y. Chen, F. He, Y. Wu, and N. Hou, "A local start search algorithm to compute exact hausdorff distance for arbitrary point sets," *Pattern Recognition*, vol. 67, pp. 139–148, 2017.

[18] M. Guthe, P. Borodin, and R. Klein, "Fast and accurate hausdorff distance calculation between meshes," 2005.

[19] D. Meagher, "Geometric modeling using octree encoding," *Computer graphics and image processing*, vol. 19, no. 2, pp. 129–147, 1982.

[20] D. P. Huttenlocher and K. Kedem, "Computing the minimum hausdorff distance for point sets under translation," in *Proceedings of the sixth annual symposium on Computational geometry*. ACM, 1990, pp. 340–349.

[21] G. Rote, "Computing the minimum hausdorff distance between two point sets on a line under translation," *Information Processing Letters*, vol. 38, no. 3, pp. 123–127, 1991.

[22] B. Li, Y. Shen, and B. Li, "A new algorithm for computing the minimum hausdorff distance between two point sets on a line under translation," *Information Processing Letters*, vol. 106, no. 2, pp. 52–58, 2008.

[23] F. Chang, Z. Chen, W. Wang, and L. Wang, "The hausdorff distance template matching algorithm based on kalman filter for target tracking," in *Automation and Logistics, 2009. ICAL'09. IEEE International Conference on*. IEEE, 2009, pp. 836–840.

[24] R. E. Kalman *et al.*, "A new approach to linear filtering and prediction problems," *Journal of basic Engineering*, vol. 82, no. 1, pp. 35–45, 1960.

[25] X.-D. Chen, W. Ma, G. Xu, and J.-C. Paul, "Computing the hausdorff distance between two b-spline curves," *Computer-Aided Design*, vol. 42, no. 12, pp. 1197–1206, 2010.

[26] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger, "The r*-tree: an efficient and robust access method for points and rectangles," in *ACM Sigmod Record*, vol. 19, no. 2. Acm, 1990, pp. 322–331.

[27] N. R. Brisaboa, S. Ladra, and G. Navarro, "k2-trees for compact web graph representation," in *International Symposium on String Processing and Information Retrieval*. Springer, 2009, pp. 18–30.

[28] H. Samet, "The quadtree and related hierarchical data structures," *ACM Computing Surveys (CSUR)*, vol. 16, no. 2, pp. 187–260, 1984.

[29] S. Alvarez-Garcia, S. Folgar, and S. Ladra, "Resumenes de grafos usando k2-trees," 2012.

[30] G. de Bernardo Roca, "New data structures and algorithms for the efficient management of large spatial datasets," Ph.D. dissertation, Universidade da Coruna, 2014.

[31] N. R. Brisaboa, S. Ladra, and G. Navarro, "Compact representation of web graphs with extended functionality," *Information Systems*, pp. 152–174, 2014.

[32] N. Roussopoulos, S. Kelley, and F. Vincent, "Nearest neighbor queries," *SIGMOD Rec.*, vol. 24, no. 2, pp. 71–79, May 1995. [Online]. Available: http://doi.acm.org/10.1145/568271.223794

[33] A. Corral, Y. Manolopoulos, Y. Theodoridis, and M. Vassilakopoulos, "Closest pair queries in spatial databases," *SIGMOD Rec.*, vol. 29, no. 2, pp. 189–200, May 2000. [Online]. Available: http://doi.acm.org/10.1145/335191.335414